设计实验室 13　　6.01 –Fall 2011

我走在路上

目标： 在这个实验中，您将实现一个系统，用于估算机器人从不确定位置沿着走廊移动时的位置。您将：

- 理解一个预处理器，它将传感器数据转换为状态估计器的输入。
- 为 6.01 机器人构建一个具有现实观测和转换模型的状态估计器和随机状态机以实现定位。在 SOAR 模拟器中构建一个完整的系统，展示机器人在一维世界中的位置定位。

资源：您可以在任何装有 SOAR 的计算机上进行此实验。此实验应与搭档一起完成。

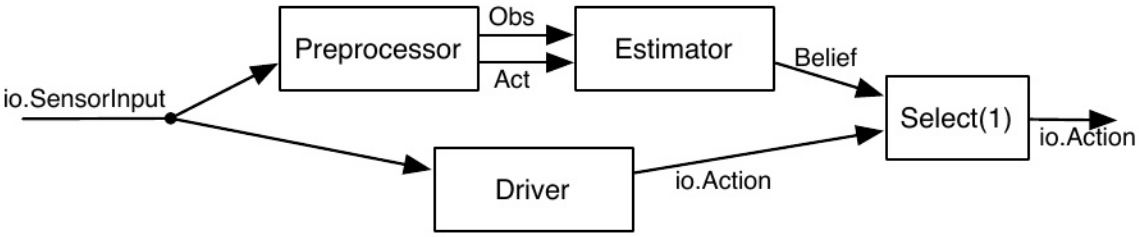
Do athrun 6.01 getFiles。相关文件（在 ~/Desktop/6.01/designLab13 中）是
· lineLocalizeSkeleton.py：用于编写您的代码的文件
· lineLocalizeBrain.py：用于运行以测试机器人定位的大脑文件

如果您尚未完成第 12 周 **2.3 部分**的内容，也未阅读**第 11 周 1.7 部分**以及**作业 4**，那么现在就去做。

1 概述

在本实验中，我们将按照第 11 周第 **1.7** 个辅导问题和第 12 周第 2.3 个辅导问题的概述，在 SOAR 中实现并最终测试一个机器人定位器。本实验将重点关注一维世界中的定位，但基本思路将使您能够在后续实验中为二维世界实现一个定位器。

以下是我们将要构建的这个系统的架构。



驱动程序、预处理器和选择状态机类已经实现。驱动程序生成 io.Action 实例，使机器人向前移动；选择（1）状态机将值的元组（或列表）作为输入，并始终返回元组的第二个元素作为输出。机器人知道它可能处于的每个可能离散位置的理想读数，但不知道它最初的位置；状态估计过程的目标

Some of the software and design labs contain the command athrun 6.01 getFiles. Please disregard this instruction; the same files are available on the 6.01 OCW Scholar site as a .zip file, labeled Code for [Design or Software Lab number].

其目的是确定机器人的位置。这种行为的效果是机器人始终向前行驶，但状态估计过程是并行运行的，并且作为一个副作用，机器人当前在世界上的位置信念状态估计值将在窗口中显示出来。

2 预处理器

预处理器模块从机器人的声纳和里程计获取传感器数据，并将其转换为有关所做的观测和所执行动作的输入信息，以供状态估计器使用。Tutor 问题 **Wk.12.2.3** 详细描述了预处理器模块。您可能需要回参考该问题中的一些定义，因此计划保持 Tutor 页面打开。

在文件 lineLocalizeSkeleton.py 中，您会发现实现预处理器模块的状态机类 PreProcess 。代码如下所示：

```
class PreProcess(sm.SM):
    def __init__(self, numObservations, stateWidth):
        self.startState = (None, None)
        self.numObservations = numObservations
        self.stateWidth = stateWidth
    def getNextValues(self, state, inp):
        (lastUpdatePose, lastUpdateSonar) = state
        currentPose = inp.odometry
        currentSonar = idealReadings.discreteSonar(inp.sonars[0],
                                                   self.numObservations)
        if lastUpdatePose == None:
            return ((currentPose, currentSonar), None)
        else:
            action = discreteAction(lastUpdatePose, currentPose,
                                    self.stateWidth)
            print (lastUpdateSonar, action)
            return ((currentPose, currentSonar), (lastUpdateSonar, action))
# Only works when headed to the right
def discreteAction(oldPose, newPose, stateWidth):
    return int(round(oldPose.distance(newPose) / stateWidth))
```

步骤 1. 确保您了解预处理器机器的实现方式。

Check Yourself 1. What is the internal state of the machine?

What is the starting state?

步骤 2. 按照如下方式对来自辅导问题 **Wk.12.2.3** 的示例进行预处理器测试：

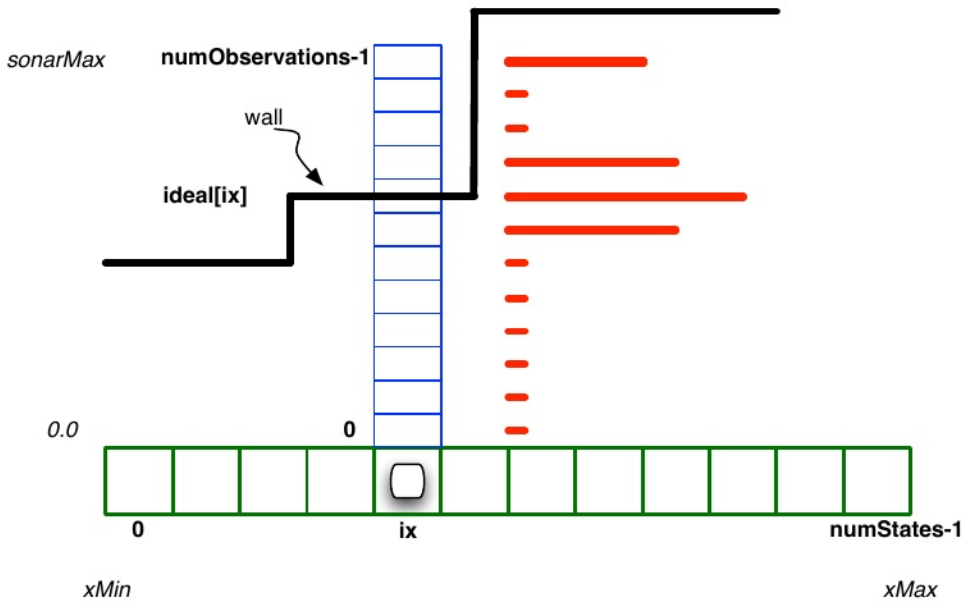
- 在 Idle 中运行 lineLocalizeSkeleton.py 文件。
- 使用与导师问题匹配的参数创建一个预处理器机器的实例，称为 pp1：10 个离散观测值、10 个离散位置值，xMin = 0.0 和 xMax = 10.0（这意味着在这个示例中状态宽度为 1.0）。
- 执行 pp1.transduce（preProcessTestData）。
- 确保输出结果与辅导题的输出结果一致。

请注意，预处理机器会在每一步打印其输出结果。

3 状态估计器

在我们的架构中，估计器模块将是 **seGraphics.StateEstimator** 的一个实例，我们已经编写了这个模块；它类似于你在 **Wk.12.2.2** 中编写的状态估计器，但它会在一对窗口中显示当前的状态估计和观测概率。每当我们创建一个状态估计器的实例时，我们必须传入一个 ssm.**StochasticSM** 的实例，该实例描述了我们对试图估计其隐藏状态的系统的了解。在本节实验中，我们的任务是为机器人定位问题创建适当的 ssm.StochasticSM，包括初始**信念分布**、**观测模型和转移**模型。我们试图估计的状态是机器人位置的*离散化* x 坐标，其范围为 0 到 numStates - 1。

您可能会发现参考以下图形会有所帮助，该图形对一个具有离散位置的机器人进行了示意性说明，该机器人观测着来自其旁边交错排列的墙壁的离散声纳读数。



加粗的标签是离散的状态和观测指标。
斜体标签是实数值的机器人位置或声纳读数。
深色线条是墙的实际位置。
绿色的方框是机器人可能的位置。
蓝色的方框可能是离散的声纳读数。
红色柱条表示分布观测值 Distribution（ix）。

文件“lineLocalizeSkeleton.py”包含以下一个过程的框架，该过程应当构建并返回相应的 ssm. StochasticSM 模型。其参数为：

- 理想值：一系列**离散化**的理想声纳读数，长度为 numStates 。
- xMin、xMax：机器人能够移动的 x 坐标的最小值和最大值
- numStates：x 范围被划分的离散状态的数量
- numObservations：离散观测值的数量

```
def makeRobotNavModel(ideal, xMin, xMax, numStates, numObservations):
    startDistribution = None
    def observationModel(ix):
        pass
    def transitionModel(a):
        pass
    return ssm.StochasticSM(startDistribution, transitionModel, observationModel)
```

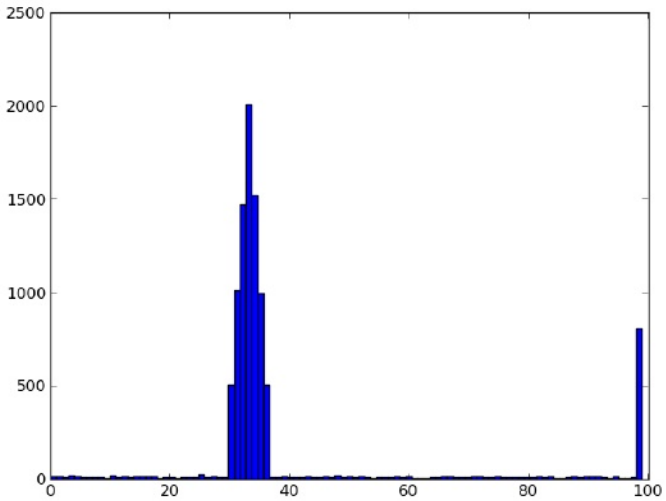
3.1 初始分布

第三步 定义 startDistribution，它应该在所有可能的离散机器人位置上均匀分布。您可以使用 dist.**squareDist（lo, hi）** 在整数范围创建均匀分布。

3.2 观测模型

观测模型是一个**条件概率分布**，表示为一个以状态（离散机器人位置）作为输入并返回可能观测值（离散声纳读数）分布的过程。我们的任务是创建一个观测模型，该模型能够描述机器人处于特定位置时可能出现的声纳读数分布。

该图展示了在一种情况下生成的 10,000 个声纳读数直方图，在这种情况下，在 0 到 1.5 米的范围内有 100 种可能的离散声纳值，并且理想的声纳读数为 0.5 米。x 轴是离散的声纳读数，y 轴是落入该区间（在 10,000 次读数中）的读数数量。



它具有以下特点:

- 由于反射等因素，总是存在非微不足道的可能性获得最大范围内的观测值。最大值为 sonarDist.sonarMax 。
- 最有可能在理想距离获得观测值，但观测中可能会有较小的相对误差（也就是说，当物体实际在 0.9 米时，我们可能看到它在 0.88 米）。
- 由于有人路过等原因，进行任何观察的可能性很小。

特别注意噪声分布的“宽度”。重要的是，编写您的混合模型时要使其对声纳读数的离散化粒度敏感:在现实世界中，对于不同的粒度，在相同数量的噪声情况下，就数量而言的宽度会有所不同。

您应该使用 dist.**MixtureDist、dist.triangleDist、dist.squareDist 和** dist.**DeltaDist** 来构建一个能很好地描述直方图中所示数据的分布。

自我检查 2. 勾勒出您的观测模型的计划。确保您了解模型的类型以及您想要创建的混合分布。如果您对其中任何一点不确定，请询问工作人员。

步骤 4. 实现观测模型并对其进行测试，以确保其合理性。它不必与图中的直方图完全匹配。

为了进行调试，您可以像这样创建一个模型（即 ssm.StochasticSM）：

```
model = makeRobotNavModel(testIdealReadings, 0.0, 10.0, 10, 10)
```

然后您可以像这样获得观测到的条件概率分布:

```
model.observationDistribution
```

在此，testIdealReadings 是来自辅导问题 **Wk.12.2.3** 的同一组理想读数。

通过绘制分布图来调试您的分布，确保您是以 -n 启动 Idle 的。如果 d 是您创建的一个分布，您可以使用 distPlot.plot(d) 对其进行绘图。

如果 observationModel 是您的观测模型，使用 testIdealReadings 中的读数，写下 observationModel(7) 中概率最高的 4 个条目（这是 DDist 的一个实例）。这里的 7 代表什么?

第 5 步。现在，为这个案例制作一个模型，使用 100 个观测区间，而不是 10 个。

```
model100 = makeRobotNavModel(testIdealReadings100, 0.0, 10.0, 10, 100)
```

绘制模型和模型100中机器人位置7的观测分布。确保它们一致且正确。

检查 1. 第 **13 周 2.1**：向一位工作人员展示您的观测分布图，并解释它们的意思。

3.3 过渡模型

转换模型是一个条件概率分布，表示为一个以一个动作作为输入并返回一个过程的过程；该过程以起始状态（离散机器人位置）作为输入，并返回对所得状态（离散机器人位置）的分布。您可以计算如果里程计没有误差将会导致的下一个位置，然后返回一个分布，该分布考虑到了机器人报告的运动中可能存在误差这一事实。

目前，在转换中唯一的误差是由于报告的动作离散化所致。思考一下，给定报告的动作是移动了 k 个离散位置，机器人可能移动到的离散位置有哪些。使用三角形分布来对离散化误差进行建模。

自我检查 3. 勾勒出您的转换模型计划。确保您了解模型的类型以及您想要创建的分布。如果您对其中任何一点不确定，请询问工作人员。

第 6 步。实现转移模型并对其进行测试，以确保其合理性。创建一个 ssm.StochasticSM 对象，然后像这样获取转移模型（这是一个返回条件概率分布的过程）：

```
model = makeRobotNavModel(testIdealReadings, 0.0, 10.0, 10, 10)
model.transitionDistribution
```

如果 transitionModel 是您的转换模型，请写下 transitionModel(2)(5)（这是 DDist 的一个实例）。这里的 2 和 5 分别代表什么？请确保结果对您来说是有意义的。

3.4 组合预处理和估计

第 7 步。现在我们将把刚刚制作的两个模块放在一起，并确保它们能正常工作。使用 sm.**Cascade** 将您的 PreProcess 类的一个实例和 **seGraphics.StateEstimator** 类的一个实例组合起来，其中给定的您的 ssm.StochasticSM 模型使用 10 个离散观测值、10 个离散位置值、xMin = 0.0 和 xMax = 10.0。将这个机器称为 ppEst 。

自我检查 4. 执行 ppEst.transduce（preProcessTestData）。将结果与**第** 12 周 2.3 节中的信念状态进行比较。请记住，您现在假设存在有噪声的观测和有噪声的动作。您的结果与导师给出的结果一致吗?

检查 2. 第 **13 周 2.2**：向一位工作人员展示您对上述问题的答案。解释它们的意思。

4 综合起来

现在，我们将把所有机器组合在一起，以实现一种能够控制机器人的行为。文件 lineLocalizeBrain.py 包含了所有必要的框架。它进行了一次调用，您需要思考一下：

```
robot.behavior = \
    lineLocalize.makeLineLocalizer(numObservations, numStates, ideal, xMin, xMax, y)
```

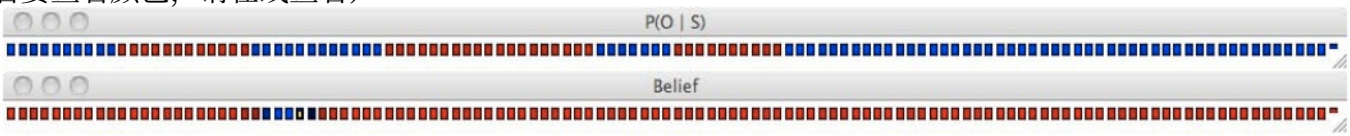
第 8 **步**。在您的 lineLocalizeSkeleton.py 文件中，使用上述所示的参数实现 makeLineLocalizer 过程；它应该构建一个完整的机器人行为，如架构图所概述的，其输入是 **io.SensorInput** 实例，输出是 **io.Action** 实例。在软件文档中阅读关于 sm.**Select** 状态机的相关内容。

您将需要预处理器和估计器机器的实例，就像您在上一个部分创建的那样，以及驱动状态机。驱动是一个状态机，其输入是 io.SensorInput 的实例，输出是 io.Action 的实例。您可以用以下方式创建它：

```
move.MoveToFixedPose(util.Pose(xMax, robotY, 0.0), maxVel = 0.5)
```

假设机器人从具有 y 坐标 robotY 的某个位置开始，并向右移动直至其 x 坐标达到 xMax 。

第 9 步。在 worlds/oneDdiff.py 中使用 lineLocalizeBrain.py 启动并运行您的行为。它将弹出类似这样的窗口（若要查看颜色，请在线查看）：



第一个窗口显示，对于每个状态（机器人可能的离散位置），当前观测值在该状态下的可能性有多大。在这个例子中，机器人当前观测到的值在它处于任何蓝色标记的位置时都可能被观测到，而在红色标记的位置不太可能被观测到。第二个窗口显示当前的信念状态，使用颜色来表示概率。黑色表示均匀概率，较亮的蓝色表示更有可能，较亮的红色表示不太可能。

不太可能。机器人的实际位置在信念状态窗口中以一个小金色的正方形显示。

使用 Soar 中的“**步**长”按钮，让机器人一步一步移动，并观察和理解显示屏上的内容。在显示屏上的内容变得有趣之前，有必要让机器人移动两步。

第 10 步。现在在 oneDreal.py 世界中运行您的行为（您需要编辑 lineLocalizeBrain.py 中选择世界文件的行，并且在 soar 中选择一个新的模拟世界）。这个世界与 oneDdiff.py 之间的本质区别是什么?

第 11 步。现在在名为 oneDslope.py 的世界中运行您的行为，不要**更改大脑中所选**的**世界**文件。这意味着机器人**认为**自己处于名为 oneDreal.py 的世界中，并具有适用于该世界的观测模型，但实际上它处于一个完全不同的世界中。运行它时会发生什么？显示的内容意味着什么?

检查 3. **第 13 周 2.3**：向一名工作人员展示您的运行定位系统。解释显示窗口中颜色的含义，并说明您的系统所做的操作是合理的。解释为什么 oneDreal 和 oneDdiff 的行为**有所**不同。解释当世界和模型之间存在不匹配时会发生什么。

5 如果你有兴趣做更多……

以下是这个实验的一些可能的扩展。

5.1 真实机器人

在真正的机器人上测试你的定位器。你需要:

- 将最大速度设置为 0.1
- 从大脑中取出离散的步长调用。
- 将 cheatPose 更改为 False
- 将驾驶员目标姿势的 y 值更改为 0.0 。
- 使用覆盖有气泡膜的盒子来搭建一个与 oneDreal.py 相对应的世界。
- （可能）调整您的声纳模型中的噪声量和运动误差。

5.2 处理传送

将此代码添加到您的脑文件:

```
teleportProb = 0.0
import random
class RandomPose:
    def draw(self):
        x = random.random()*(xMax - xMin) + xMin
        return (x, y, 0.0)
io.enableTeleportation(teleportProb, RandomPose())
```

如果您将 teleportProb 设置为大于 0 的值，那么它将以该概率在每个移动步骤中将机器人“传送到”从机器人的 x 范围中均匀随机选择的 x 坐标处。（保持相同的航向和 y 坐标）。这是测试您定位能力的一种好方法。

如有必要，修改您的转移分布，使其能够应对机器人可能会瞬移的世界。思考一下您的模型中哪些参数应该与瞬移概率匹配。

提高传送概率，看看你的机器人能否应对。